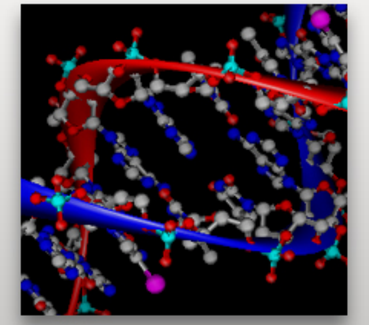
Developing Reliable and High-Performance Drivers for Mac OS X

Session 505

These are confidential sessions—please refrain from filming or taking pictures.



OS Foundations

Developing Reliable and High-Performance Drivers for Mac OS X

Dean Reece
IOKit Team Manager

http://developer.apple.com/wwdc2005/

What You'll Learn

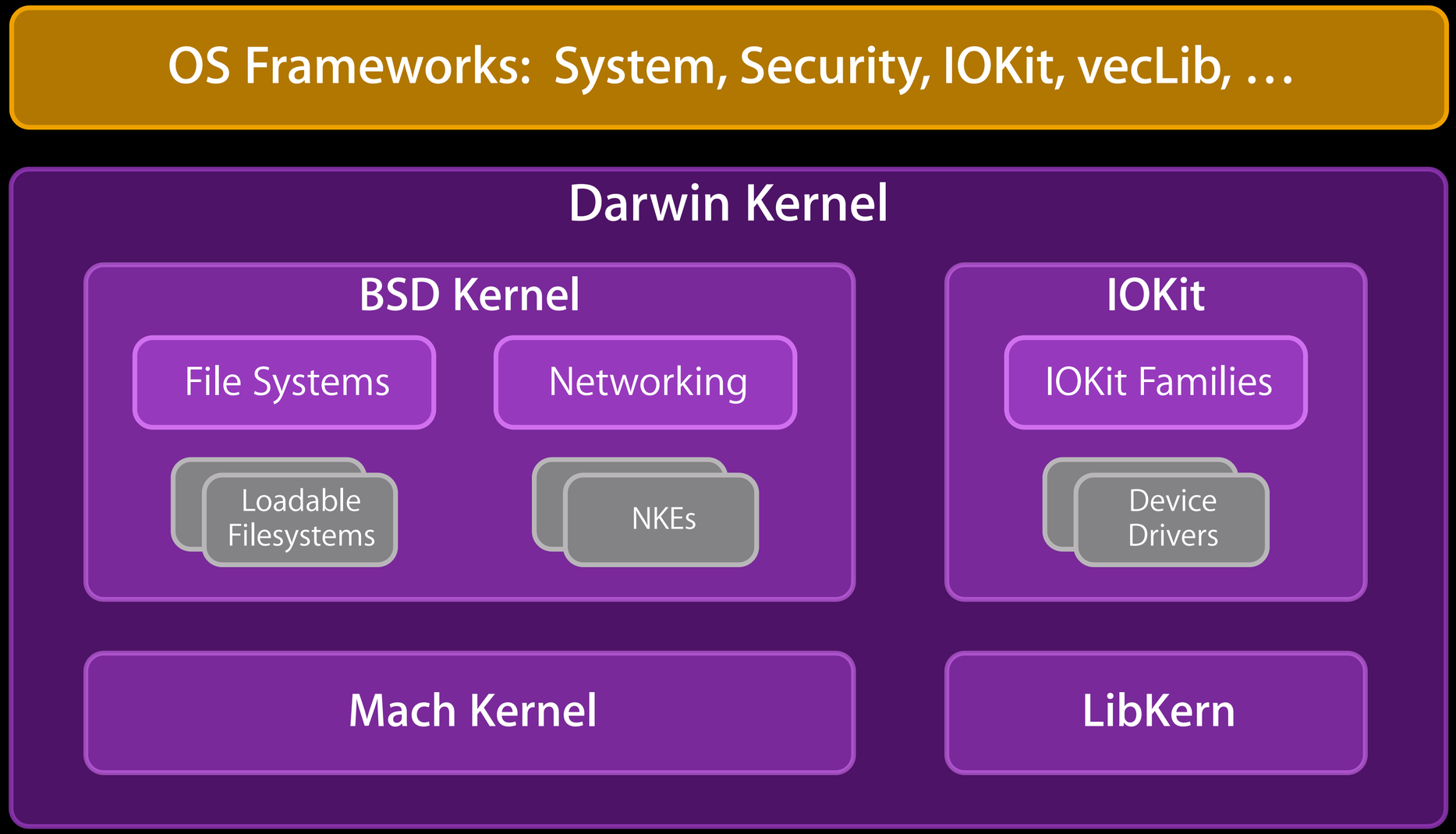
- **Developing for Binary Compatibility**
  Tools, VTable patching, KPIs, spanning releases

- **KEXT Debugging**
  Tools, techniques

- **Developing for Performance**
  Memory, threading, and tools

- **Developing for Reliability**
  Common pitfalls, testing techniques and tools

Technology Framework



KEXT Binary Compatibility

Future-proofing your KEXT

http://developer.apple.com/wwdc2005/

KEXT Management in Tiger

- Tiger KEXT Management is largely unchanged from Panther
  - Improved KEXT cache invalidation logic
- After installing, updating, or removing a KEXT:
  `sudo touch /System/Library/Extensions`

  Do *not* manually remove or update the KEXT cache files
- You must restart the system for the changes to take effect if you are:
  - Updating a driver that is already loaded and can't be unloaded
  - Or installing on a release prior to 10.3
- You may use the new driver *without requiring a restart* if you are:
  - Installing a new driver, or updating a driver that is not loaded
  - And installing on a 10.3 or later

Refer to Technical Q&A QA1319 *"Installing a KEXT Without Rebooting"*

KPIs in Tiger

- Biggest KEXT-level change for developers is introduction of KPIs
  - IOKit and LibKern are mostly unaffected
  - The bulk of KPI changes effect BSD and Mach interfaces
- KEXTs must choose either the new KPIs or the Compatibility interface
  - Mixing the two namespaces is not supported
  - New KPIs available under `com.apple.kpi.*` (v8) identifiers
  - Compatibility interface available under `com.apple.kernel.*` (v1–v7.9.9)

Related session

Performance & Tools Lab: Writing File Systems Using Supported KPIs

Lab
Wednesday 3:30pm

FAQ

One KEXT Across Multiple Releases?

Question

With KPIs, how do I release one KEXT for multiple OS Releases?

Answer

If your kext will run on Tiger using the compatibility (<8.0) interfaces, then you can ship a single KEXT

If it fails to load, then you will probably need to provide a pre-Tiger KEXT and a Tiger KEXT that uses the new KPIs

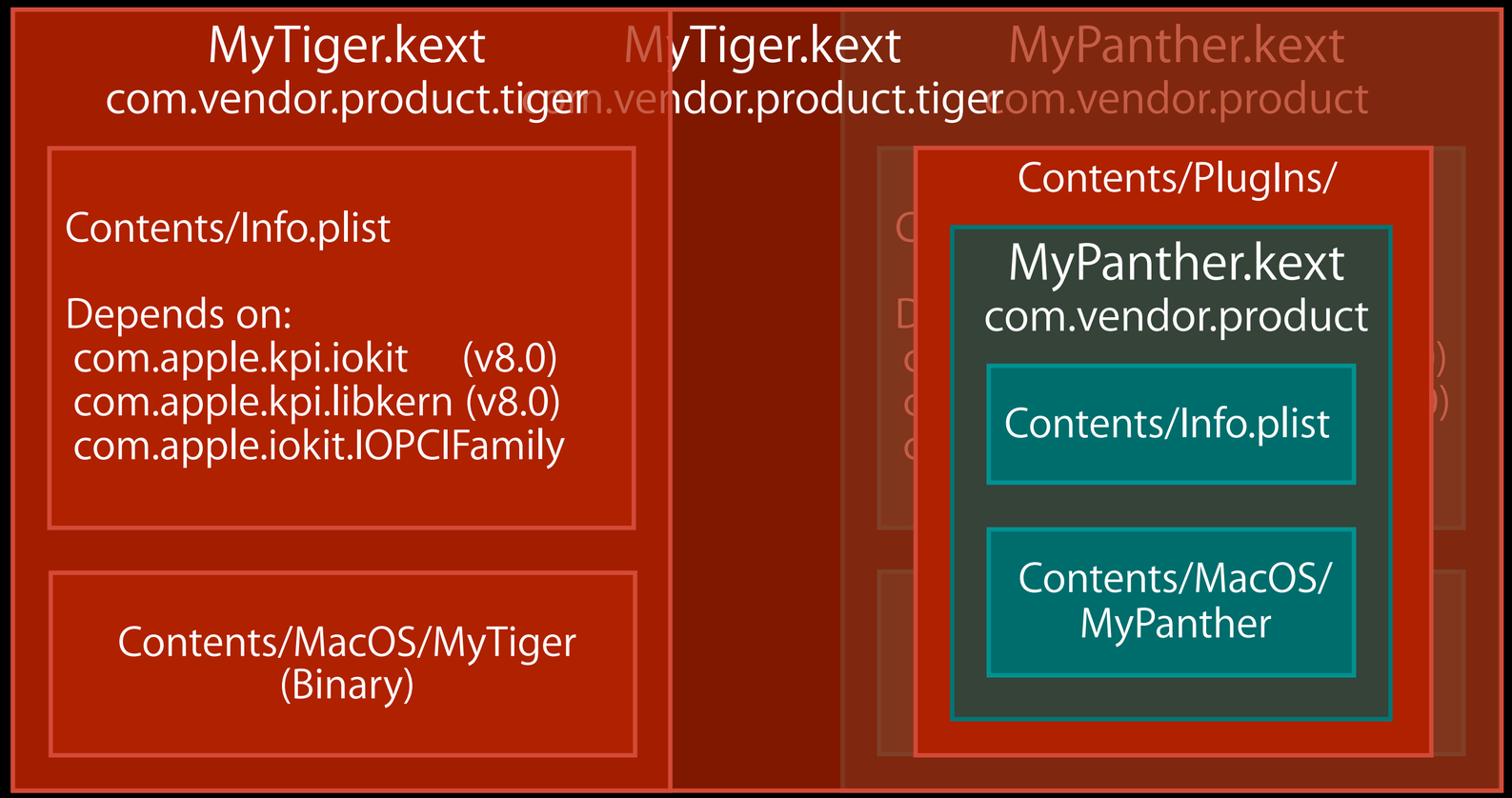
FAQ

One KEXT Across Multiple Releases?

- If you need to deliver two KEXTs
  - Don’t just install the one for the user’s current OS since this will break across an upgrade
  - Consider nesting one KEXT inside the PlugIns folder of the other KEXT
    Nesting is cosmetic and will not effect matching or dependencies
  - Use separate CFBundleIdentifiers for the two KEXTs
  - Dependencies must be such that only one KEXT can load on any given OS

FAQ

One KEXT Across Multiple Releases?



KEXT Binary Compatibility

- KEXT Backward Compatibility
  - The ability to load and use an existing KEXT on a newer OS release
  - IOKit provides Backward Compatibility for KEXTs across OS releases
- KEXT Forward Compatibility
  - The ability to load and use a new KEXT on an older OS release
  - IOKit does not provide Forward Compatibility for KEXTs
- IOKit does not guarantee Source Compatibility between releases, though we avoid breaking it without good reason

KEXT Binary Compatibility

- Which Compiler to use?
  - You may use gcc3.3 or gcc4.0 to build KEXTs for 10.2.0 and later releases
  - Do *not* use gcc2.95 for new KEXTs—it is supported only for backward compatibility for kexts built against 10.1 or 10.0

- Build your KEXT using the SDK for the oldest OS release you support

- You must use the `OSDefine...()` and `OSDeclare...()` macros for every class you implement that inherits from outside of your KEXT
  - Otherwise, when the next OS is released, you'll see:
    "`<your_class> is not compatible with its superclass, <your_superclass> superclass changed?`"
  - `IOService::` changed in Tiger, so building your kext on Panther and loading it on Tiger is a good test for proper use of these macros

Building KEXTs with gcc4.0

- gcc4.0 is the Default Compiler for Tiger Developer Tools
- gcc4.0 is more strict with C++ code, so you may encounter source compatibility issues using gcc4.0 for IOKit KEXTs
- One common problem is that casting a method to a function pointer is not legal in gcc4.0
  - Use `OSMemberFunctionCast()` to convert a member function into a function pointer suitable for callback routines
  - Eliminates the need for static stub functions
  - No longer need the `-fpermissive` flag for casting
- If you are not able to use gcc4.0, you can switch to gcc3.3:
  `sudo /usr/sbin/gcc_select 3.3`

KEXT Debugging

One machine, or two?

http://developer.apple.com/wwdc2005/

Single Machine Debugging

- Not recommended
- Slow cycle time if reboots are required
  - Make sure your driver can be unloaded/reloaded safely
- You will panic your test system
  - Risky since disk may be corrupted during a panic
  - Backup anything important, such as development source
- No local kernel debugger support

Single Machine Debugging: *Logging using IOLog()*

- Can be useful to figure out general code flow and resolve loading and initialization issues

- Intended as a logging service, not a debug interface
  - Limited buffer size (4K) and bandwidth
  - Log file accumulates and fills disk
  - Not synchronous by default

- Switch to debugging mode by setting `bootarg io=0x200000`
  - Makes output to console synchronous
  - Additional log messages may be enabled by setting additional bits
  - See `IOKit/IOKitDebug.h` for details
  - Log in using user name “>console” to get into text console

Single Machine Debugging: *New in Tiger*

- Two new KPIs in libkern/OSDebug.h to programmatically request a stack backtrace
  - `OSReportWithBacktrace(const char *str, ...)`
    - Generates a 4-level backtrace similar to a `panic()`, but returns
    - Handy to figure out how you got into a function
  - `OSBacktrace(void **bt, unsigned count)`
    - The core implementation of `OSReportWithBacktrace()`
    - Doesn't log anything—fills in an array with addresses
    - Handy for more complex condition detection and filtering
- These functions are not fast—do *not* leave in production KEXTs

Single Machine Debugging: *Decoding a Panic*

- Collect the backtrace
  - Get Panic log from /Library/Logs/panic.log
  - Use `OSReportWithBacktrace()` and collect BT from /var/log/system.log
- Download the appropriate Kernel Debug Kit from ADC
- Construct kext symbols for all kexts indicated in the log
- Run gdb on the appropriate symbol'd kernel and add each symbol'd kext
- Use `x/i <addr>` on each address in the panic log
  - Start with PC and LR, then examine the stack
  - Remember to subtract 4 from each stack address
  - Use `c++filt` utility to convert mangled symbols into C++ symbols
    `__ZNK9IOService11getWorkLoopEv` ⇒ `IOService::getWorkLoop() const`

Refer to Technical Note TN2063 *"Understanding and Debugging Kernel Panics"*

Two Machine Debugging

- Recommended for all Kernel-level development
  - Much faster development cycle
  - Target machine is expendable—Development system is safe
- Use gdb to remotely control and monitor kernel execution
  - Provides much more flexibility than decoding a backtrace
  - Live examination of memory, structures, and objects
  - Breakpoints, single stepping, etc.
- Three ways to attach to target machine
  - Via ethernet using built-in gdb support
  - Via FireWire using /Developer/Extras/Kernel Debugging
    - Install AppleFireWireKDP.kext on target machine
    - Run FireWireKDPProxy on debugger machine
  - Via Core Dump

Two Machine Debugging: *Remote kprintf()*

- Debug print mechanism is kprintf()
- Synchronous
- Output is disabled unless enabled by debug= 0x08 bootarg
- Two ways to connect
  - Via Serial port (if available)
  - Via FireWire (coming soon to FireWire SDK)
- Similar to local logging, but with advantages
  - Session can be scrolled, searched, and saved
  - FireWire logging is much faster than local printf() or serial kprintf()

Two Machine Debugging: *Kernel Core Dumps*

- One machine can be a collection server for many client systems
- Clients are configured to send a core file on every panic
- Takes a little work to set up, but tremendously powerful
- Allows failed machine to be rebooted prior to analysis
- Core file can be kept with problem report for convenient handling
- Open core files in gdb and perform any memory examination operation
  - Very much like being attached to a paniced machine
  - Of course, no execution (breakpoints, single stepping) is possible

Refer to Technical Note TN2118 *"Kernel Core Dumps"*

KEXT Performance

Getting more with less

http://developer.apple.com/wwdc2005/

What Is Performance?

Throughput?

- Optimize for peak or sustained events-per-second
- Most commonly considered important in server environments or bulk data handling such as A/V data stream manipulation

Latency?

- Optimize for minimal delay from input to output
- Most important in A/V mixing and capture/playback
- Low “jitter” is often considered as important as low latency

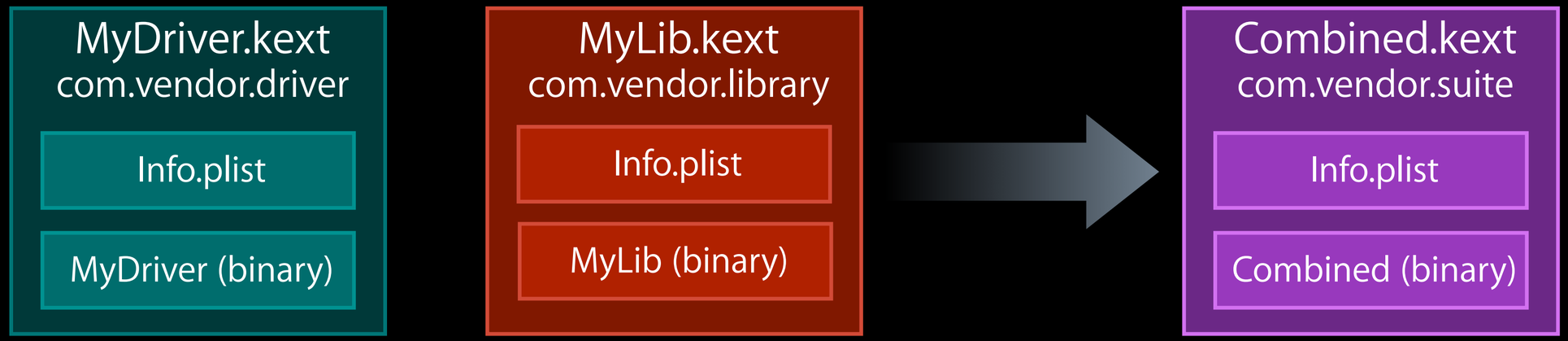
Efficiency!

- Optimize for low CPU cost per event
- Often results in better throughput and/or latency
- Generally reduces footprint and increases battery life
- Helps the system run cooler and more quietly

Memory Footprint: *Wired Memory*

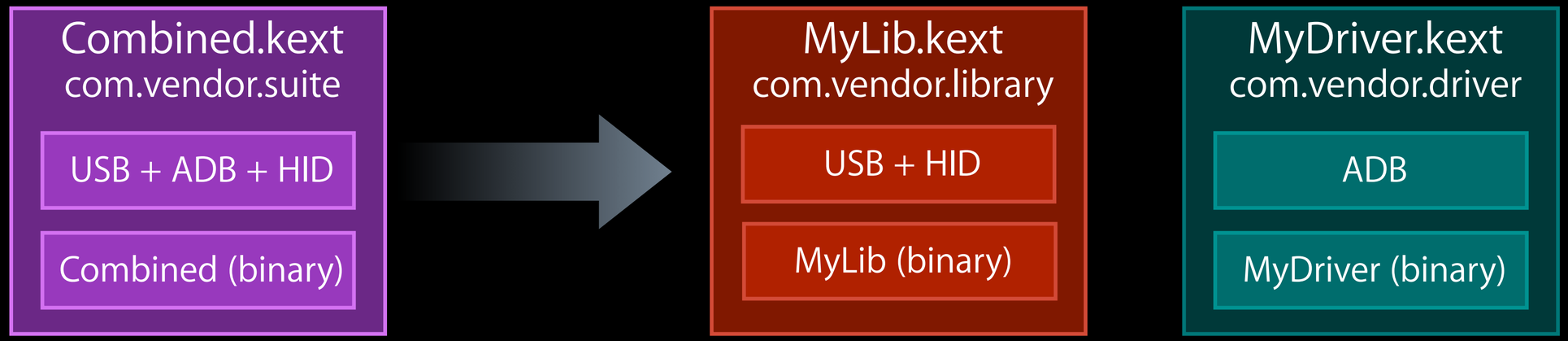
- Wired Memory is the most expensive kind of memory
  - In-kernel allocations are wired memory by default
  - All text and data associated with a KEXT are loaded into wired memory
- Each IOService subclass has ≈2K Wired Memory "Tax"
  - Use classes, but don't create superfluous classes
  - Subclass from the highest-level class that is practical
  - For internal helper classes, consider using your own root class
- Each KEXT has a ≈4K Wired Memory "Tax" when loaded
  - In multi-kext suites, combine KEXTs that are nearly always used together
  - For KEXTs that match against nubs from different families, split "rare" interfaces into separate bus-specific kexts

Memory Footprint: *Wired Memory*



In multi-kext suites, combine KEXTs that are nearly always used together

Memory Footprint: *Wired Memory*



For KEXTs that match against nubs from different families,
split “rare” interfaces into separate bus-specific kexts

Memory Footprint: *Keep Only What You Need*

- Don’t put large blocks of data in the IORegistry or IOPersonalities
  - Only put data there that is necessary for operation
  - Make debug data contingent on “Development” build style
- Don’t build firmware data into your driver
  - Use a helper app to pass the data down to the KEXT on demand
  - Put firmware data into a separate KEXT; match it in when needed; unload it when done
  - Consider moving init-only code into a separate KEXT
- Don’t fight IOKit when porting a driver to Mac OS X
  - Make use of inherited functionality—remove code where possible
  - Follow IOKit design patterns instead of surrounding existing code with “impedance matching” code

Memory Footprint: *Fragmentation*

- Think of the Kernel as a long-running app—no exit to clean-up
- Fragmentation increases the wired memory footprint
- Avoid lots of small allocations where possible
  - Use the stack for small (dozens of bytes) working buffers
  - Consider grabbing whole pages for short-use memory
- Cluster long-term allocations as much as possible
  - These are the allocations that pin pages down
  - Consider using a pool arrangement similar to IOCommandPool

Memory Footprint: *Leaks*

- A Kernel or KEXT memory leak is one of the most serious bugs that frequently gets past development testing and QA
  - Wired memory leaks don't go cold and get paged out like app-level leaks
  - System will slow and eventually need to be rebooted
  - Seriously damages the User Experience of the whole platform
- Use cycling scripts to watch for leaks
  - Identify which paths cause alloc/free or retain/release operations
  - Devise cyclers which will exercise these code paths
  - Identify which classes you manage and watch them using `ioclasscount` on each cycle
  - Use `ioalloccount` to watch for non-class allocation leaks on each cycle (only on single-processor systems)
  - Use `zprint` utility to watch kalloc zone allocations

Threading and Locking

- Locks and other atomic operations are expensive on MP systems
  - Causes data cache invalidation
  - Steals time from all processors
  - Kernel functions that potentially block are doing locking or atomic ops
  - Obviously they are to be avoided on the main data path
- Context switches are expensive (relative to simple function calls)
  - Requires significant memory access as well as locking operations
  - Susceptible to ping-pong effect on MP systems (kills cache efficiency)
- Disabling interrupts to protect driver data from ISR
  - Interrupt-time processing is tricky to get right (can't block) and isn't scheduled, causing jitter
  - Disabling interrupts to protect data also causes jitter

Threading and Locking: *IOWorkLoop*

- `IOWorkLoop` is specifically designed to address these issues
  - Designed with systemic performance in mind
  - This eliminates context switches and most locking costs along the common data-movement paths
  - `IOWorkLoop` and `IOInterruptEventSource` eliminate interrupt-time processing time, allowing the scheduler to decide what needs the CPU (not the PIC)
  - Work loop inheritance is generally along vertical stacks that follow data-flow call-graphs
- Trust IOKit, particularly `IOCommandGate`, and avoid introducing addional locking unless absolutely necessary
  - Coding “gated” functions may seem unnatural at first, but the technique becomes easy and leads to a “gated community” programming style
  - Gated functions significantly aid in debugging deadlocks since they appear on gdb “showallstacks”

Where Does the Time Go?

- Use the `top` utility to get a feel for normal system utilization
- Then, run cyclers or stress tests and compare top's results to the baseline
- Kernel and all drivers are in PID 0

```
Processes:  54 total, 2 running, 52 sleeping... 174 threads            00:52:01
Load Avg:  0.28, 0.33, 0.27     CPU usage:  6.3% user, 3.6% sys, 90.0% idle
SharedLibs: num =  188, resident = 45.5M code, 4.86M data, 7.16M LinkEdit
MemRegions: num =  8712, resident =  263M + 10.1M private,  142M shared
PhysMem:   135M wired,  349M active,  290M inactive,  775M used, 1.24G free
VM: 8.43G +  125M   49301(0) pageins, 2421(0) pageouts

  PID COMMAND      %CPU   TIME    #TH #PRTS #MREGS RPRVT  RSHRD  RSIZE  VSIZE
  798 top          9.3%  0:01.31   1    18    22  1.06M   396K  2.82M+ 26.9M
  771 bash         0.0%  0:00.01   1    14    16   204K   852K   832K  27.1M
  770 login        0.0%  0:00.01   1    16    36   144K   412K   556K  26.9M
  768 Terminal     0.7%  0:02.26   4    90   131  2.01M  11.9M  8.74M   359M
  ...
    0 kernel_tas   0.9%  9:53.93  41     2  3510  36.4M     0K   116M  1.42G
```

Where Does the Time Go?

- Use Shark to profile your driver under load
- Shark.app is part of the CHUD package on the Developer DVD
- Shark profiles the entire system, including all kernel and KEXT activity (except when interrupts are disabled)
- How much CPU is your driver using when it is idle, or under load?
- You can identify hot code paths through your driver that would most benefit from optimization

Shark Resources:

- User's Guide is installed in:
  `/Developer/ADC Reference Library/documentation/CHUD`
- Make sure to install the appropriate Kernel Debug Kit SDK from:
  http://developer.apple.com/sdk

KEXT Reliability

Testing tips and common pitfalls

http://developer.apple.com/wwdc2005/

Common Pitfalls: *IORegistry Corruption*

- Unprotected access to the IORegistry is a common source of panics
  - All changes to property tables in the IORegistry must be done via the IORegistry object itself to ensure proper locking
  - Direct access to the underlying containers is not safe and will cause occasional, hard to diagnose panics (typically data access exception)
  - Modifying the contents of an OSData object is acceptable without protection

- In Tiger, you can enable strict checking by using:

```
sysctl -w debug.iokit=0x10000
```

  - We could not enable this in the release because too many drivers failed
  - We may enable this on future releases, so please find and fix any unsafe IORegistry access

Common Pitfalls: *IORegistry Corruption*

- When you find unsafe IORegistry access:
  - For simple top-level property changes use `IORegistryEntry::setProperty()`
  - For complex changes
    - Copy container to be modified
    - Make changes
    - Install changes using `IORegistryEntry::setProperty()`

- New KPIs in Tiger to help solve this problem
  - To make a suitable copy of a collection tree, use

    `OSCollection::copyCollection()`
  - To execute a function synchronized against IORegistry changes, use

    `IORegistryEntry::runPropertAction()`

Testing Tips: *Cyclers*

- Identify actions that cause activity in your driver
  - Bookends like kextload/unload, and open/close are potential cycler targets
  - Constructs that can be overwhelmed such as queues and pools

- Construct cyclers targeted at each identified path
  - Cycler issues operations at a controllable rate
  - Cycler assess driver health at suitable points in every cycle
    - Particularly look at memory use and class allocations
  - Cycler logs on each pass to help analysis in case of failure

- Apple has used cyclers to great benefit in house

Testing Tips: *KEXT Source Checklist*

- Lots of KEXTs ship with easily avoidable problems
- Make sure your classes are named using the reverse DNS idiom to avoid polluting the class namespace
- Make sure `OSDefine…()` and `OSDeclare…()` macros are correctly set up for every class you implement that inherits from outside of your KEXT
- Make sure kprintf() statements are compiled out, as they introduce a small cost even when no output is configured
- Remove or disable debugging code, particularly calls to:
  - `panic()`, `assert()`, `IOLog()`, and `OSReportWithBacktrace()`

Testing Tips: *KEXT Binary Checklist*

- Lots of KEXTs ship with easily avoidable problems
- Use `kextload -tn <name>.kext` to check for basic bundle issues
- Use `find <name>.kext -ls` and look for garbage files like `.DS_Store`
- Verify that binary has been stripped using `strip -S`
- Examine Info.plist
  - Make sure `IOKitDebug` property is removed or set to `0`
  - Make sure version numbers match everywhere
  - Update Copyright String
- Check `/var/log/system.log` for lingering debug messages

Testing Tips: *kextpert (coming soon)*

- Kextpert is a utility that checks for many common KEXT problems
- Intent is to automate detection of any problems where possible
- Generates a per-file report with four types of comments
  - Errors: Fatal errors that must be fixed
  - Warnings: Issues to investigate—they may be errors-in-hiding
  - Suggestions: Opportunitites to improve your KEXT
  - Info: Details that indicate how kextpert's analysis is going (off by default)
- Report also contains a references section to locate on-line resources to help you understand and correct issues
- Kextpert is a work-in-progress
  - Future plans include dependency analysis and OS release compatibility check
  - Coming soon—Watch the ADC website

More Information

http://developer.apple.com/wwdc2005/

Documentation, Sample Code, and Other Resources

http://developer.apple.com/wwdc2005/

Links and Mail Lists

Apple Developer Connection	
Main page	developer.apple.com
Developer mail lists	lists.apple.com
ATA and SCSI development	ata-scsi-dev@lists.apple.com
FireWire development	firewire@lists.apple.com
USB development	usb@lists.apple.com

Darwin	
At ADC	developer.apple.com/darwin
At OpenDarwin	www.opendarwin.org
Darwin mail lists	lists.apple.com
General development	darwin-dev@lists.com
FireWire development	usb@lists.apple.com
Kernel development	darwin-kernel@lists.apple.com

Related Sessions

Customizing Mac OS X Using Open Source	Session 508 Wednesday 3:30PM
Creating Drivers for USB Devices	Session 509 Wednesday 3:30PM
Core OS Enhancements for BSD Developers	Session 510 Thursday 3:30PM
Performance & Tools Lab: Writing File Systems Using Supported KPIs	Lab Wednesday 3:30PM
Performance & Tools Lab: Kernel Extensions	Lab Thursday 2:00PM

Who to Contact

Craig Keithley
I/O Technology Evangelist
keithley@apple.com

Dean Reece
IOKit Team Manager
dean.reece@apple.com

Q&A

Craig Keithley
I/O Technology Evangelist
keithley@apple.com

http://developer.apple.com/wwdc2005/

WWDC2005

Worldwide Developers Conference